XT3 Architecture and Software

NLCF User's Meeting Feb. 14, 2006

Bronson Messer

Scientific Computing Group
National Center for Computational Sciences
Oak Ridge National Laboratory

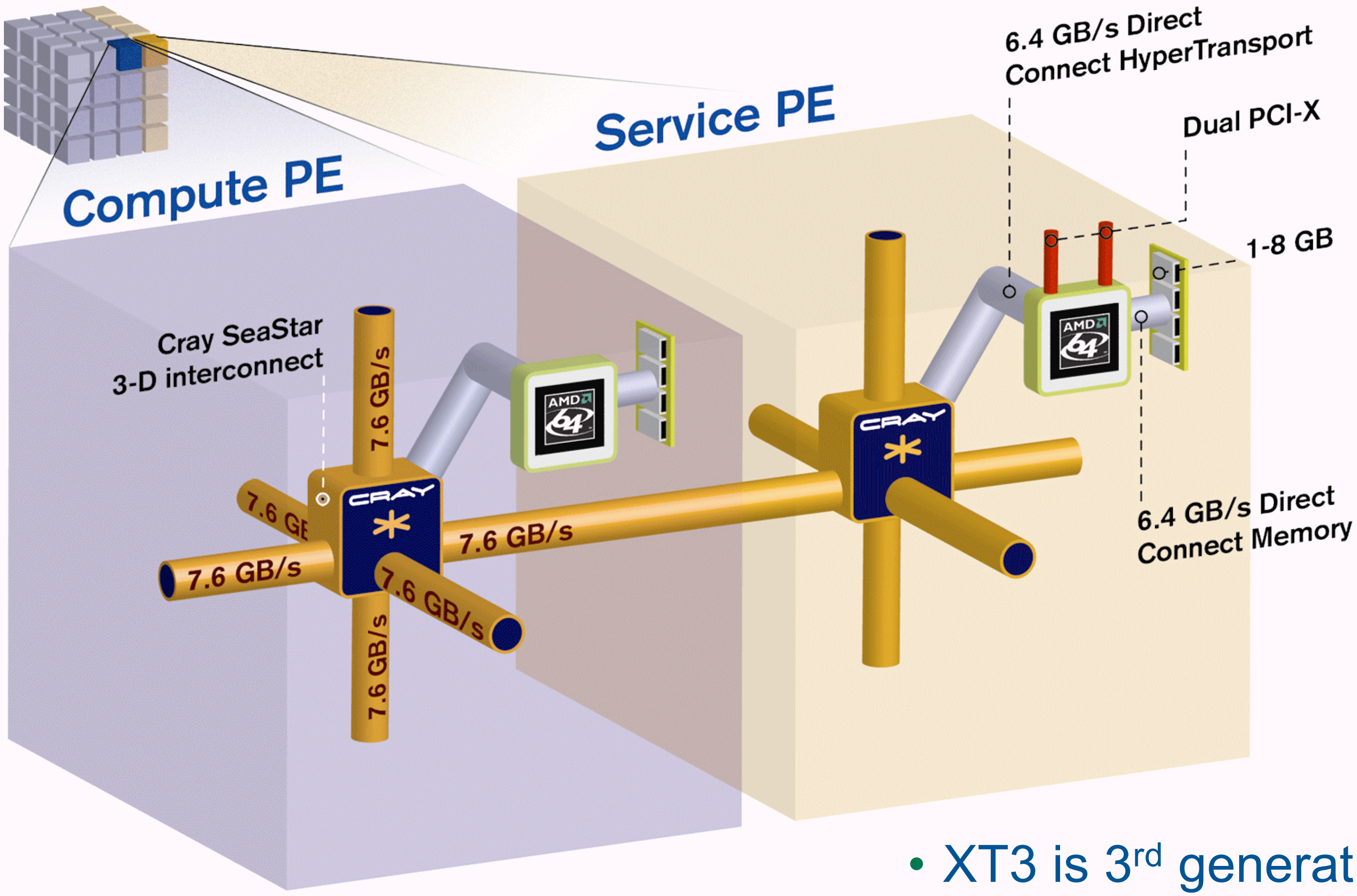
NATIONAL CENTER
FOR COMPUTATIONAL SCIENCES

OAK RIDGE
National Laboratory

jaguar (Cray XT3)

- 56 cabinets, 5212 compute PEs, 82 service PEs.
- PEs: 2.4 GHz AMD Opteron 150 connected via HT to a custom ASIC (Cray SeaStar)
- 4 PEs/compute node >10 TB RAM (2GB/PE)
- PE topology: 14x16x24 (torus in X,Z; mesh in Y)

Cray XT3 Architecture



- XT3 is 3rd generation Cray MPP
- Service nodes run Linux
- Compute nodes run Catamount quintessential kernel (qk)

catamount

- Latest in a sequence of lightweight kernel operating systems developed at Sandia and the University of New Mexico

- Scalability and performance predictability (elimination of ‘jitter’) provided by each a kernel running only one single-threaded process

- Services like paged memory, threading, TCP/IP, forks, etc. are unavailable

Current software environment

- PGI 6.0.5
- gcc 3.3
- Login nodes have kernel 2.4.21
- Unicos/lc 1.3.14
- XT/MPT 1.3.14
- acml 2.7

Customizable through modules

modules

- Lots of modules available on jaguar
- `module swap` worth remembering
- `module initadd` available, but requires a bit of scripting to make it act as most would wish
- Watch for the occasional information message when executing `module add`

compilers

- `ftn, cc,` and `CC` are very tidy wrappers for catamount compiling & linking.
- Use the wrappers essentially all the time.
  - most of your builds will be cross-compiles for catamount
  - `-target=catamount` will suppress litany of warnings
- What's different under Catamount?
  - No threads, no sockets, no fork, no dynamic libs
  - No `system()`calls
  - Catamount `malloc` is designed for large, semi-static blocks of memory; use `-lgmalloc` to get glibc `malloc`

compiling

- `-r8` to do ubiquitous scientific computing promotion
- `-g` to get debugging symbols
  - put `-g` FIRST (it implies `-O0`)
  - `-Ktrap=fp` to trap floating point exceptions, and thereby actually do useful debugging
- `-mcmodel=medium` to get > 1GB(!) if you have that much or more statically allocated storage
  - PGI memory map sets aside the other 1GB in the small memory model for stacks, shared libs, etc.

compiling (cont.)

- Try some vectorization (SSE,SSE2)
  - `-fastsse`
  - Sets optimization level to `-O2`
  - Only buys you 1 extra flop/clock for `REAL*8`, but fewer instructions are generated
  - `-Mcache_align`: if you vectorize a subroutine, but don't use `-fastsse` to build main, makes sure arrays are on cache line boundaries (part of `-fastsse`)
- Let the compiler unroll small loops
  - e.g. -Munroll=c:4 unrolls loops 4 times
- `-tp k8-64` explicitly sets optimization for 64-bit Opteron

compiling (cont.)

- `-Mprof=func` provides DWARF hooks for profiling (more later and even later…)
  - Very important if you have subroutines in Fortran modules
- `-Mipa=fast` is usually a good thing for C++
  - Make sure to put it on the link line too
- Got start-up/tabular date in binary files?
  - You may need `-byteswapio`

XT3, Altix

X1E, POWER, T3E

Cray MPICH & shmem

- Cray MPI-2 *derived* from MPICH2
  - Most important: no spawning, no thread safe
  - no `MPI_LONG_DOUBLE` type
  - Using `INTEGER*8` array sizes can cause failure

- shmem
  - `-lsma` on the link line
  - No atomic memory operations

Both are implemented with Portals low-level communication layer

MPI environment variables

You may need to (re)set a couple of MPI environment variables

- `MPICH_PTL_OTHER_EVENTS` - sets the number of events in queue to receive “all other” types of messages (i.e. a lot, e.g. `MPI_ALL_TO_ALL`)
  - Default = 2048
  - 4096 works for some codes to go to 5000 procs

- `MPICH_PTL_UNEX_EVENTS` - number of unexpected point-to-point messages (`MPI_GATHERV`)
  - Default = 20480
  - Experience shows may need to be set to 80000 or more

- `MPICH_UNEX_BUFFER_SIZE` - size of buffers for unexpected receives
  - Default = 60M
  - >400M?

Running

- PBS Pro is the batch scheduler
- You need to include `#PBS -A <identifier>` (Bobby says so! Ask your PI for your `<identifier>`.)
- Submit with `qsub <batch-script>`

- yod launches applications on compute nodes
  - `yod -sz n <executable>`
  - `yod -np n <executable>`
  - `yod -size n <executable>`

Running (cont.)

- `-small_pages` option to `yod`
  - Opteron TLB provides 512 entries for 4kB pages, or 8 entries for 2MB pages.
  - By default, Catamount uses 2MB pages
  - This allows 16MB to be mapped in the TLB (vs 2MB for 4kB pages)
  - If your code jumps around to more than 8 places in memory (e.g. you have some sort of gather/scatter loop), you may want to try `-small_pages`
- Watch your job with `xtshowmesh` or `xtshowcabs`
- `qstat -a` to check on queue status

Libraries

- Scientific
  - ACML (AMD Core Math Library)
    - BLAS, LAPACK, 1-D FFT
    - Fast intrinsics and vector intrinsics
    - `-i8` can break this! (just like MPICH)
    - LAPACK timing routines have been hacked
    - Has been compiled with `-fastsse,` so use `-Mcache_align`
  - Cray LibSci
    - ScaLAPACK, BLACS, SuperLU
- Both of these are in the default module set

Libraries (cont.)

- I/O
  - HDF5
  - Parallel and serial versions available as modules (`hdf5/1.6.4_ser` & `hdf5/1.6.4_par`)
  - Need to add link and include info to build
    - `${HDF5_FLIB}` and `${HDF5_CLIB}`
    - These also point to szip and libz

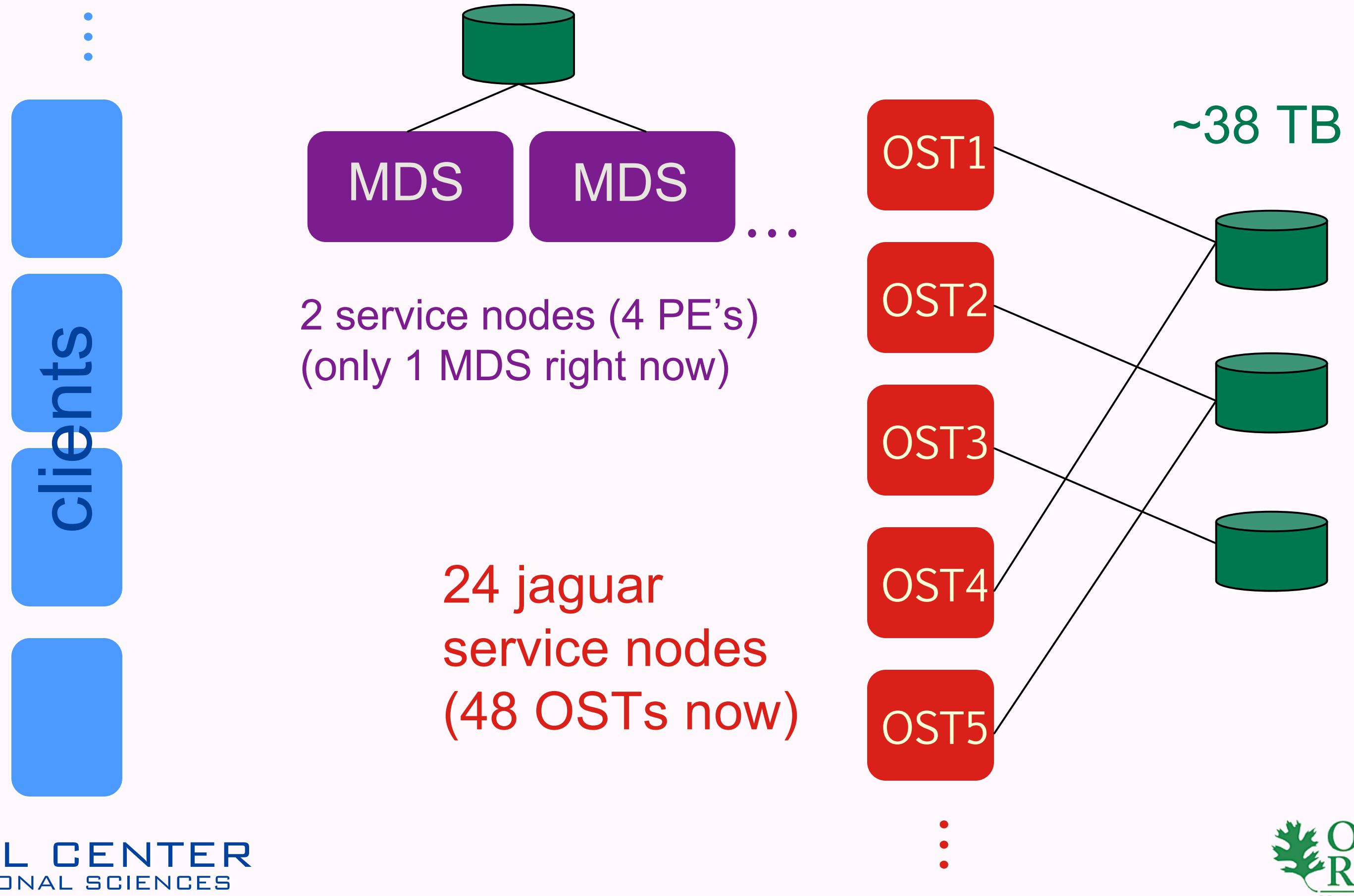
libs (cont.)

- netCDF
  - Not quite ready as a module
  - Preliminary version available in `/apps/netcdf/3.6.0/xt3_pgi605/`
  - Any need for pnetCDF?
- fftw
  - A module exists (double precision only)
  - Please let us know what you need

lustre

- A parallel, object-based filesystem which aggregates a number of storage servers together to form a single coherent file system that can be accessed by a client system.

jaguar nodes (compute)



2 service nodes (4 PE’s) (only 1 MDS right now)

24 jaguar service nodes (48 OSTs now)

More on Lustre

- The only way to do I/O on the compute nodes *without going back out through the yod* (and thereby throttling I/O pretty well) is via liblustre.
- The lustre module is currently loaded by default: linked in when you build a catamount executable.

Striping

- You can change the striping pattern across the OSTs on a *per directory* basis yourself
- You should have a good understanding of *how and how much* your application outputs before you attempt this!
  - YOU CAN FILL UP INDIVIDUAL OSTs!
  - Do not stripe your work directory wholesale!

CAUTION
THIS SIGN HAS
SHARP EDGES
DO NOT TOUCH THE EDGES OF THIS SIGN
ALSO, THE BRIDGE IS OUT AHEAD

Striping (cont.)

- You should think of this as “preparing the ground.”
  - The striping obtains the next time you write to the directory/write a file
  - If you change the settings for an existing directory, you will need to copy the files elsewhere and then copy them back to inherit the new settings.
- Striping is probably most beneficial when the application writes all the data to one file, either by collection or direct access.

Striping (cont.)

- `lfs` gets/sets striping information
- `lfs getstripe <file>` will tell you the striping information for a file
- `lfs find -v <dir/file>` equivalent

- `lfs setstripe <dir>` *size start number*
  - `lfs setstripe <dir>` *0 -1 -1* means no striping

Other I/O stuff

- Buffering `stdio`
  - all that `stdout` from write(*,*) goes through the yod, and, by default, `stdio` is unbuffered
  - This translates to about 10 bytes/s
  - Call setvbuf() in C, setvbuf3f(lu, type,size) in Fortran
- You can’t use named pipes under Catamount

Performance & profiling

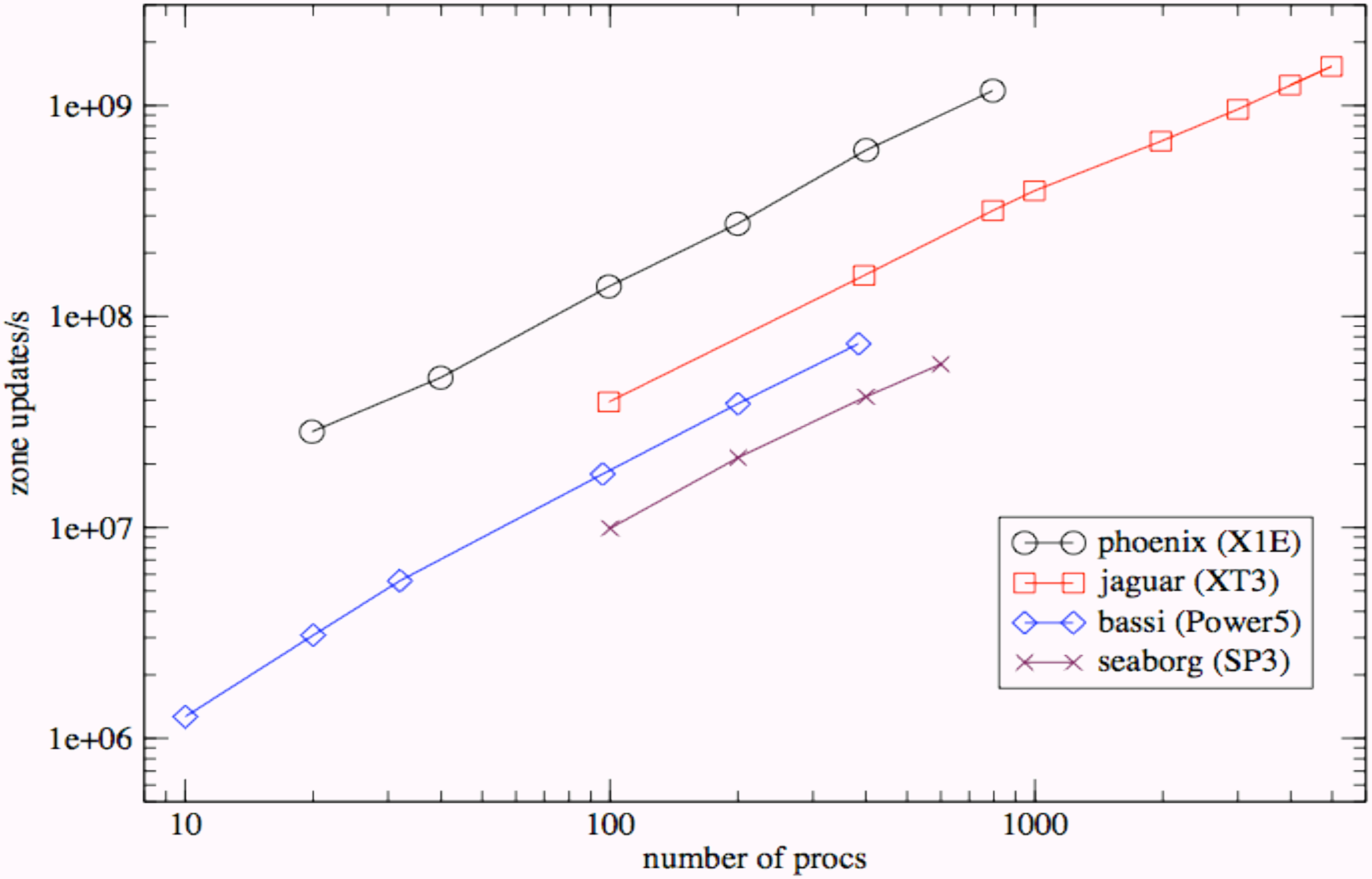
- PAPI available as a module
- Catamount ostensibly makes ‘elapsed time’ and ‘cpu time’ the same
  - You can’t use `clock`, `etime`, or `times`
  - You can use Fortran intrinsic `cpu_time()` and `mpi_wtime()`
  - `dclock()` too, but uses uncalibrated CPU frequency
  - `getrusage()` returns user time = sys time and totaltime = 2*(user time)
- CrayPat tutorial on Thursday
  - `pat_hwpc` is simple way to get flop rate, cache misses, etc. for whole code (no instrumentation, but load the craypat module before compiling)

More profiling, etc

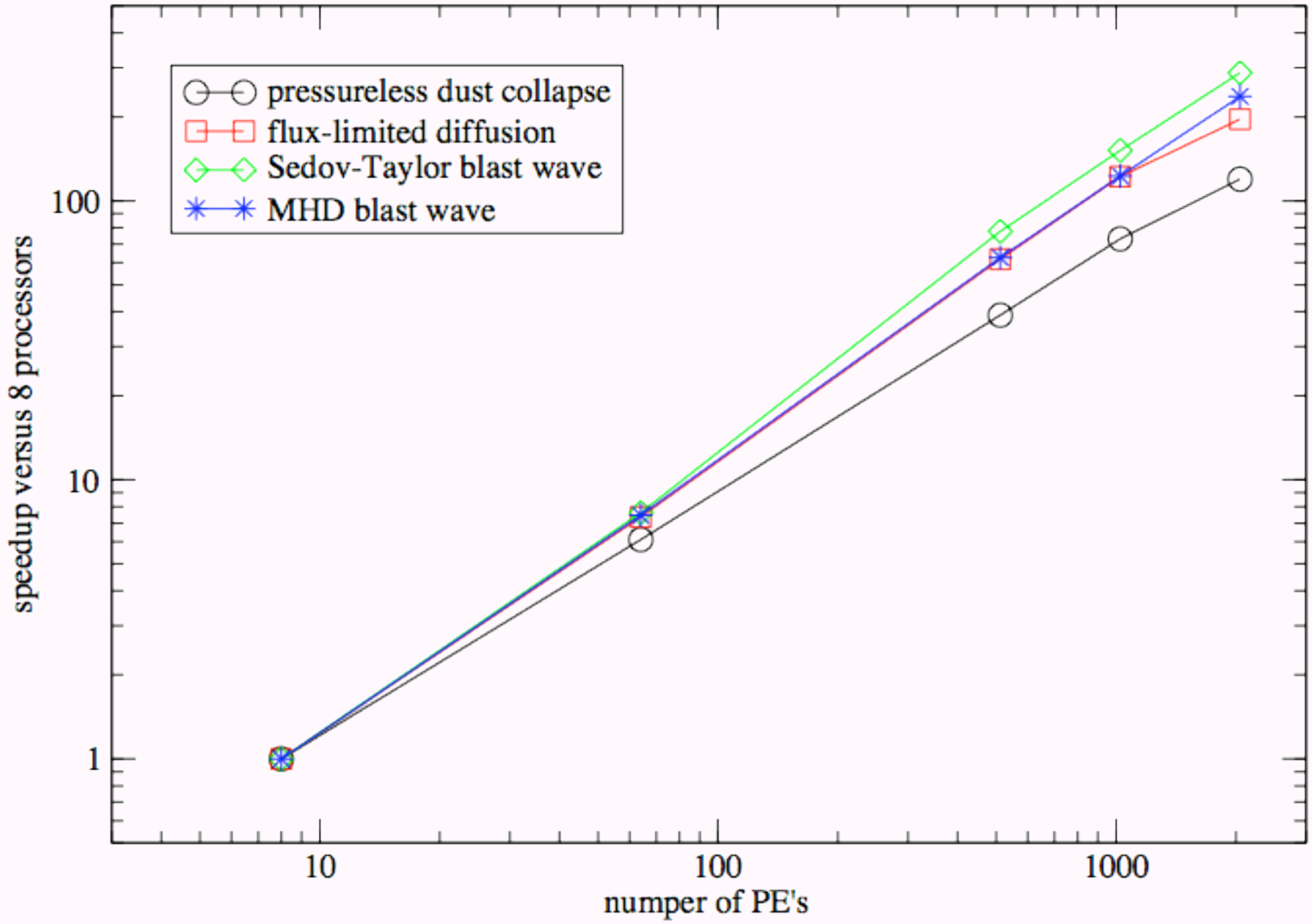
- How to check stack and heap usage
  - Examples to check stack and heap usage on http://info.nccs.gov/resources/jaguar/faq
  - In ~jlbeck/xt3/heapmax
    - heapmax.o file contains function dumpheap. A user can add call dumpheap() anywhere in their source and it will dump the same heap information that they would get on exit when this is compiled in.
- Do you have subroutines in Fortran modules?
  - Did you remember `–Mprof=func`

Benchmarks

VH-1 (astro)



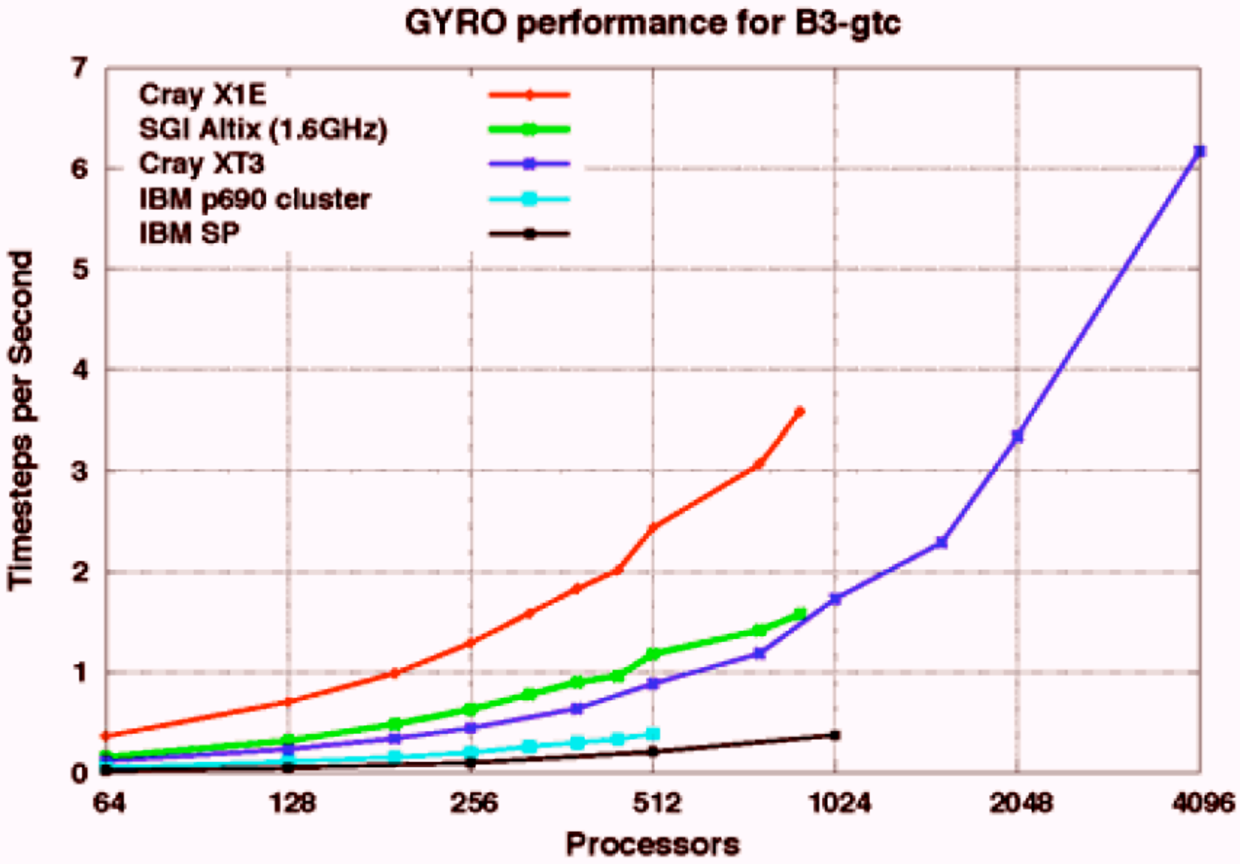
ZEUS-MP (astro)



NATIONAL CENTER
FOR COMPUTATIONAL SCIENCES

OAK RIDGE
National Laboratory

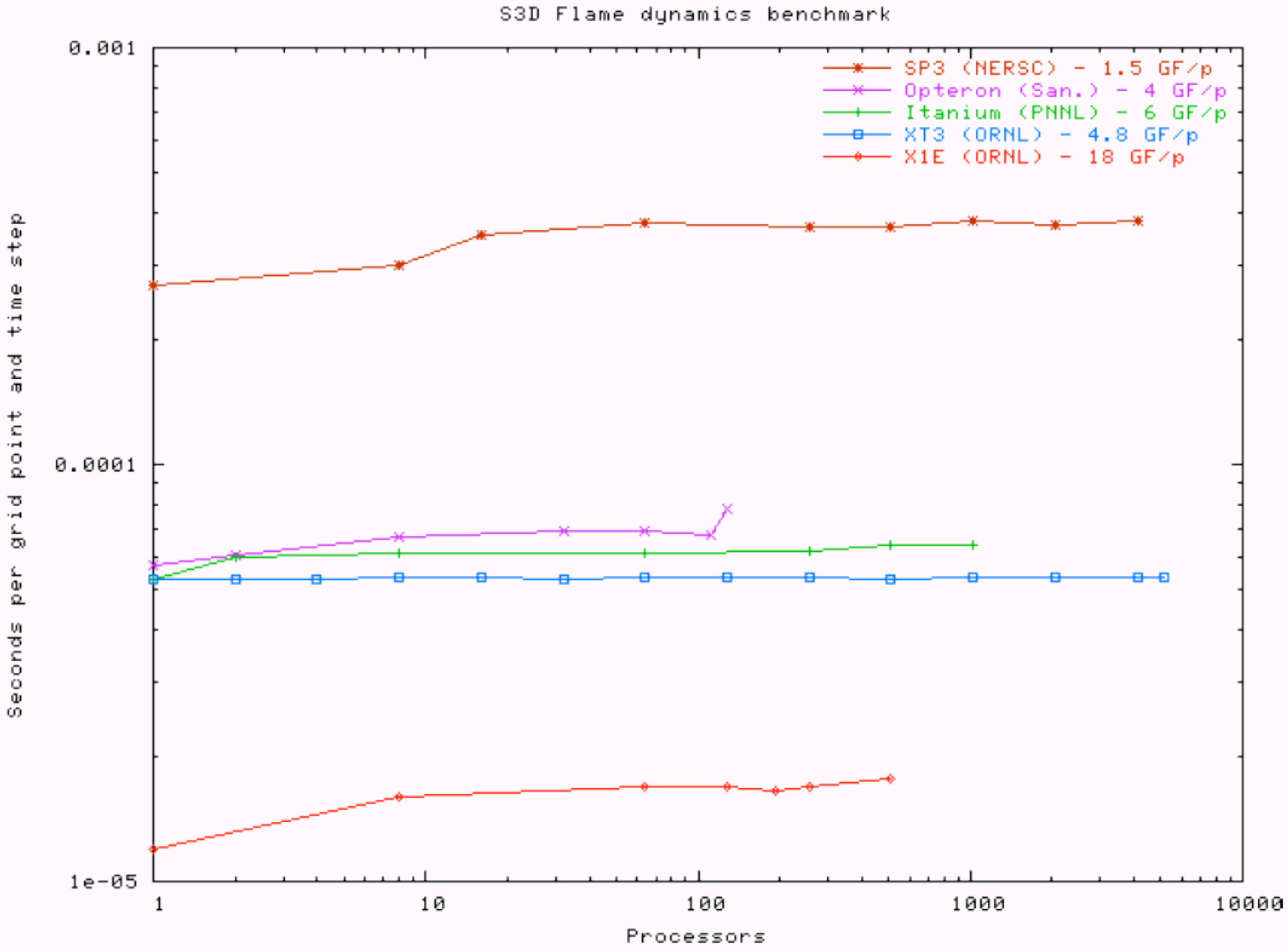
GYRO (fusion)



NATIONAL CENTER FOR COMPUTATIONAL SCIENCES

OAK RIDGE National Laboratory

S3D (combustion)



faster

jaguar

phoenix

More Information

- http://info.nccs.gov/ has a large amount of information on all systems, including jaguar, and is continuously updated
- http://docs.cray.com/ has remarkably readable documentation
- Contact us (help@nccs.gov) and we will obviate the problem one way or the other (answer, fix, help you fix, squall to Cray…)